Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks

Arnaud Casteigts, Serge Chaumette, Afonso Ferreira

Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks*

Arnaud Casteigts[1], Serge Chaumette[2], and Afonso Ferreira[3]**

[1] SITE, University of Ottawa,
casteig@site.uottawa.ca
[2] LaBRI, Université de Bordeaux,
serge.chaumette@labri.fr
[3] CNRS - MASCOTTE Project, INRIA Sophia Antipolis,
afonso.ferreira@sophia.inria.fr

Abstract. Besides the complexity in time or in number of messages, a common approach for analyzing distributed algorithms is to look at their assumptions on the underlying network. This paper focuses on the study of such assumptions in dynamic networks, where the connectivity is expected to change, predictably or not, during the execution. Our main contribution is a theoretical framework dedicated to such analysis. By combining several existing components (local computations, graph relabellings, and evolving graphs), this framework allows to express detailed properties on the network dynamics and to prove that a given property is necessary, or sufficient, for the success of an algorithm. Consequences of this work include (i) the possibility to compare distributed algorithms on the basis of their topological requirements, (ii) the elaboration of a formal classification of dynamic networks with respect to these properties, and (iii) the possibility to check automatically whether a network trace belongs to one of the classes, and consequently to know which algorithm should run on it.

Key words: Dynamic networks, distributed algorithms, evolving graphs, local interactions, topological assumptions.

1 Introduction

The past decade has seen a considerable research effort devoted to the design of distributed algorithms and protocols targeting dynamic network topologies. It appears, however, that most of the assumptions considered when examining algorithm requirements still relate to static properties such as the *size*, *density*, or *geometry* of the target network. Assumptions that really relate to the network dynamics, when any, are generally stated using non-formal expressions, such as

* Partially supported by A.N.R. grant No ANR-05-SSIA-0002-01.

** A.Ferreira is currently on leave as Head of Science Operations at the COST Office, Brussels, BE. http://www.cost.esf.org.

"*nodes are expected to move slowly enough to..*", or "*nodes cannot leave the network for a long time*", or by assuming a given *mobility model* (whose concrete topological implications remain unclear). The dynamic properties highlighted in such a way make the fair comparison of algorithm requirements rather complicated and often ambiguous.

Our work aims at providing general formalisms and methods for studying fundamental properties of dynamic networks, and more particularly their impact on distributed systems. As an illustrative example, let us consider the broadcasting of an information within the dynamic network depicted by Figure 1. The possibility to complete the broadcast in this scenario clearly depends on which node is the initial emitter: a and b may succeed, while c cannot. Why? How can we formulate this intuitive property that the topology evolution must have with regard to the emitter and other nodes? How can we formally prove it as a necessary condition to obtain broadcast completion? While rather simple, such a characterization might be difficult to obtain with usual graph formalisms and computation models.

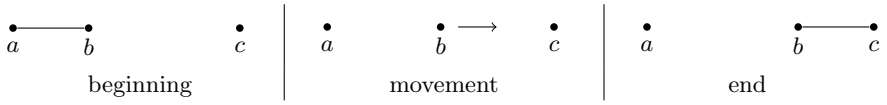


Fig. 1. A basic dynamic scenario, where a node (b) moves during the execution.

This paper introduces a theoretical framework dedicated to such kind of analyses. This framework is intended to serve as a general basis for studying fundamental properties of distributed algorithms in dynamic networks. Contrary to the work in [AAD+06], where the authors first make a strong topological assumption (all pairs of nodes repeatedly meet during the execution) and then characterize the solvable problems in this context, we consider the exact opposite approach by studying, for a given solution (*i.e.*, algorithm), the *necessary* and/or *sufficient* conditions it requires on the topology. To the best of our knowledge, this is the first attempt in such a direction.

The strength of the proposed framework lies in its basic components, which are an appropriate combinatorial model to represent dynamic topologies (*evolving graphs* [Fer04]), and a very high-level interaction model to describe distributed operations (*local computations*, with the associated formalism of *graph relabellings* [LMS99]). The next section is devoted to the presentation of these existing components. In Section 3 we combine them to set up the new analysis framework. This framework is then applied in Section 4 to the analysis of three basic algorithms (one propagation and two enumeration algorithms), whose intuitively apparent properties are here formally characterized. Based on the analysis results, Section 5 shows how algorithms can be compared on the basis of their topological requirements, and reciprocally how dynamic networks can be classified according to the algorithms they support. Finally, we discuss

the possibility to check automatically the inclusion of a given network trace to one of the classes. Section 6 concludes with some avenues for further research.

2 Related work

We describe here the formalisms and theoretical tools that compose the proposed analysis framework: *Local Computations* to abstract the communication model, *Graph Relabelling Systems* to describe local computation algorithms, and *Evolving Graphs* to express properties on dynamic topologies. Their comprehension is required to ensure a clear understanding of the following sections, where they are combined together.

2.1 Abstracting communications through local computations and graph relabellings

Distributed algorithms can be expressed using a variety of communication models (*e.g.* mailbox, shared memory, and message passing). Whereas a vast majority of algorithms is designed in one of these models (predominantly the message passing model), the very fact that one of them is chosen implies that the obtained results (*e.g.* positive or negative characterizations and associated proofs) are limited to the scope of this model. This problem of diversity among formalisms and results, already pointed out twenty years ago in [Lyn89], led researchers to consider higher abstractions when studying fundamental properties of distributed systems.

Local computations and *Graph relabellings* were jointly proposed in this perspective in [LMS99]. These theoretical tools allow to represent a distributed algorithm as a set of local interaction rules that are independent from the effective communications. Within the formalism of graph relabellings, the network is represented by graph whose vertices and edges are associated with labels that represent the algorithmic state of the corresponding nodes and links. An interaction rule is then defined as a transition pattern $(preconditions, actions)$, where *preconditions* and *actions* relate to the label values. Since these interactions are local, each transition pattern must involve a limited and connected subset of vertices and edges. Figure 2 shows different scopes for the transition patterns, which are not necessarily the same for *preconditions* and *actions*.

More formally, let the network topology be represented by a finite undirected loopless graph $G = (V_G, E_G)$, with V_G representing the set of nodes and E_G representing the set of communication links between them. Two vertices u and v are said *neighbors* if and only if they share a common edge $\{u, v\}$ in E_G. Let $\lambda : V_G \cup E_G \rightarrow \mathcal{L}^*$ be a mapping that associates every vertex and edge from G with one or several labels from an alphabet $\mathcal{L}$ (which denotes all the possible states these elements can take). The state of a given vertex v, *resp.* edge e, at a given time t is thus denoted by $\lambda_t(v)$, *resp.* $\lambda_t(e)$. The whole *labelled graph* is represented by the pair (G, λ), noted $\mathbf{G}$.

Fig. 2. Different powers of local computations; the scope of *preconditions* is depicted in *white (on left sides)*, while the scope of *actions* is depicted in *black (on right sides)*. The *dashed elements* represent entities (vertices or edges) that are considered by preconditions but remain unaffected by actions. The reader is referred to [CMZ06] for a comparative study of some of these models.

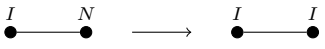
According to [LMS99], a complete algorithm can be given by a triplet $\{\mathcal{L}, \mathcal{I}, P\}$, where $\mathcal{I}$ is the set of initial states, and P is a set of *relabelling rules* (transition patterns) representing the distributed interactions. The Algorithm 1 below ($\mathcal{A}_1$ for short), gives the example of a one-rule algorithm that represents the general broadcasting scheme discussed in the introduction. We assume here that the label I (*resp.* N) stands for the state `informed` (*resp.* `non-informed`). Propagating the information thus consists in repeating this single rule, starting from the emitter vertex, until all vertices are labelled I.[4]

Algorithm 1 A propagation algorithm coded by a single relabelling rule (r_1).

initial states: $\{I, N\}$ (I for the initial emitter, N for all other vertices)
alphabet: $\{I, N\}$

preconditions(r_1): $\lambda(v_0) = I \wedge \lambda(v_1) = N$
actions(r_1): $\lambda(v_1) := I$

graphical notation :



Remark 1. Although the three algorithm examples provided in this paper consider pairwise interactions (and more specifically the model depicted on Figure 2(c)), the concepts and discussions developed in Sections 3 and 5 are not dedicated to it. Note that models such as those of Fig. 2(b) and 2(b) reflect well a wireless computing environment where nodes update their states according to those of their neighbors.

Regarding the organization of collaborations between nodes, it is important to note that the algorithm specification does not stipulate how the nodes must collaborate, *i.e.,* the way they select each other to perform a common computation step. From the abstraction level of local computations, this underlying synchronization is seen as an implementation choice, which implies that local computation algorithms may not be deterministic at this level. As discussed later on, characterizing *sufficient* conditions will require additional assumptions on this underlying layer (which is not the case for *necessary* conditions).

[4] Detecting such a final state is not part of the given algorithm. The reader interested in termination detection as a distributed problem is referred to [GMMS02].

2.2 Expressing dynamic network properties using Evolving Graphs

In a different context, *evolving graphs* [Fer04] have been proposed as a combinatorial model for dynamic networks. The initial purpose of this model was to provide a suitable representation of *fixed schedule dynamic networks* (FSDNs), in order to compute optimized communication schemes such as shortest, fastest and foremost paths. In such a context, the evolution of the network was known beforehand. In the present work, we propose to use evolving graphs for a different purpose, namely to express topological properties in dynamic networks. It is important to keep in mind that the analyzed algorithms are never supposed to know the evolution of the network ahead of time.

An evolving graph is a structure in which the changing connectivity of a dynamic network is recorded (see Figure 3). More formally, let $\mathcal{S}_\mathbb{T} = t_0, t_1, ..., t_n$ be a sequence of dates in $\mathbb{T}$ (usually $\mathbb{R}^+$). Except for t_0 and t_n, all these dates correspond to a topological event that modifies the network. Let $\mathcal{S}_G = G_0, G_1, ..., G_{n-1}$ be the corresponding sequence of graphs, with each G_i the graph corresponding to the period $[t_i, t_{i+1}[$. Finally, let us denote by G (alone) the union graph of all G_i, called the *underlying graph*. Then the triplet $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_\mathbb{T})$ is the corresponding *evolving graph*. As shown in Figure 3, this graph can be represented by the underlying graph G whose edges and vertices (only edges here) are associated with their presence interval indices. Henceforth, we will use the notations $V_\mathcal{G}$ and $E_\mathcal{G}$ to denote $V(G)$ and $E(G)$, the sets of all vertices and edges that exist at some point of the network life. Note that whereas used as *undirected* in this paper, evolving graphs were initially introduced as *directed*, and considered also bandwidth restrictions on edges, which is not used here.

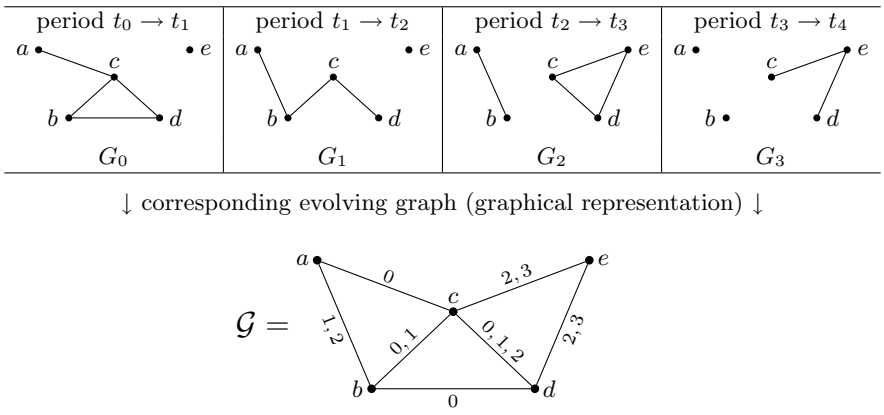


Fig. 3. Example of an evolving graph covering a period of time from t_0 to t_4

Further definitions on evolving graphs (given an evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_\mathbb{T})$).
Predecessor of a date: for any date d in $\mathbb{T}_{[t_i, t_{i+1}[}$, with $t_i, t_{i+1} \in \mathcal{S}_\mathbb{T}$, we say that t_i is the *predecessor* of d in $\mathcal{S}_\mathbb{T}$, and we note $pred(d) = t_i$.

Journey: given a sequence of couples $\mathcal{J} = \{(e_1, \sigma_1), ..., (e_i, \sigma_i), ..., (e_k, \sigma_k)\}$ composed of edges from $E_{\mathcal{G}}$ and dates from the continuous domain $\mathbb{T}$, $\mathcal{J}$ is called a *journey* if and only if $\sigma_1, \sigma_2, ..., \sigma_k$ is non-decreasing and for all i in $1..k$, $e_i \in E(G_{pred(\sigma_i)})$, that is, the edge e_i exists at time σ_i. Less formally, a journey can be thought of as a path *over-time* from one vertex to another. A journey from a vertex u to a vertex v is noted $\mathcal{J}_{(u,v)}$.
Discrete Journey: a *discrete journey* is a journey so that every date of the sequence $\sigma_1, ..., \sigma_k$ is in $\mathcal{S}_{\mathbb{T}}$, instead of $\mathbb{T}$. It allows to represent in a single entity all the possible journeys occurring on the same sequence of edges during the same sequence of intervals. This also allows to consider such entities as *subgraphs* of the evolving graph $\mathcal{G}$, and to note $\mathcal{J} \subseteq \mathcal{G}$. The point is that every normal journey $\{(e_1, \sigma_1), ..., (e_i, \sigma_i), ..., (e_k, \sigma_k)\}$ can be associated with a discrete journey $\{(e_1, pred(\sigma_1)), ..., (e_i, pred(\sigma_i)), ..., (e_k, pred(\sigma_k))\} \subseteq \mathcal{G}$, and every discrete journey implies an infinity of normal journeys for the corresponding edges and intervals.
Strictness of a discrete journey: a discrete journey is said *strict*, noted $\mathcal{J}_{strict}$, if its sequence of dates $\sigma_1, \sigma_2, ..., \sigma_k$ is strictly increasing.

To give a few examples on the graph of Figure 3,
-$J_{(a,e)}$=$\{(ac, \sigma_1 \in [t_0, t_1[), (ce, \sigma_2 \in [t_2, t_3[)\}$ is a *normal* journey from a to e ;
-$J_{strict(a,e)}$=$\{(ac, 0), (ce, 2)\}$ is a *discrete* (and strict) journey from a to e ;
-$J_{(a,e)}$=$\{(ac, 0), (cd, 0), (de, 3)\}$ is a discrete (non-strict) journey from a to e ;
-$J_{strict(a,e)}$=$\{(ac, 0), (cd, 1), (de, 3)\}$ is a discrete (and strict) journey from a to e.

Note that journeys are naturally oriented, in the sense that a journey from one vertex to another does not imply the existence of a journey in the reverse direction (*e.g.* from e to a). From this point on, unless said explicitly, we will only consider discrete journeys, and denote them by the sole term *journey*.

3 The proposed analysis framework

As a recall of the previous section, the algorithmic state of the network is given by a labelling on the corresponding graph G, then noted $\mathbf{G}$. As another recall, we denote by G_i the graph covering the period $[t_i, t_{i+1}[$ in the evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$, with $G_i \in \mathcal{S}_G$ and $t_i, t_{i+1} \in \mathcal{S}_{\mathcal{T}}$. Note that the notation G was used here with two different meanings: the first as the generic letter to represent the network, the second to denote the *underlying graph* of $\mathcal{G}$. Both notations are kept in the following, while preventing the text from ambiguities.

3.1 Putting the pieces together: relabellings over evolving graphs

For an evolving graph $\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$ and a given date index $i \mid t_i \in \mathcal{S}_{\mathbb{T}}$, we denote by $\mathbf{G}_i$ the labelled graph $(G_i, \lambda_{t_i+\epsilon})$ representing the network state just after the topological event of date t_i, and by $\mathbf{G}_{i[}$ the labelled graph $(G_{i-1}, \lambda_{t_i-\epsilon})$ representing the network state just before it. We note,

$$Event_{t_i}(\mathbf{G}_{i[}) = \mathbf{G}_i \ .$$

A number of distributed operations can occur between two consecutive events. Hence, for a given algorithm $\mathcal{A}$ and two consecutive dates $t_i, t_{i+1} \in \mathcal{S}_\mathcal{T}$, we denote by $\mathcal{R}_{\mathcal{A}_{[t_i,t_{i+1}[}}$ the relabelling sequence induced by $\mathcal{A}$ on the graph G_i during the period $[t_i, t_{i+1}[$, and have,

$$\mathcal{R}_{\mathcal{A}_{[t_i,t_{i+1}[}}(\mathbf{G}_i) \;=\; \mathbf{G}_{i+1[} \;.$$

For the sake of simplicity, we authorize the notation $r_i(u,v) \in \mathcal{R}_{\mathcal{A}_{[t,t'[}}$ to denote the fact that a rule r_i is applied on the edge (u,v) during $[t,t'[$. A complete execution sequence from t_0 to t_{last} is given by an alternated sequence of relabelling steps and topological events, noted,

$$X{=}\mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$$

The combined formalism is summed up on Figure 4. As mentioned at the end of Section 2.1, the execution of a local computation algorithm is not necessarily deterministic, and may depend on the way nodes select one another at a lower level. Hence, we denote by $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm $\mathcal{A}$ over an evolving graph $\mathcal{G}$.

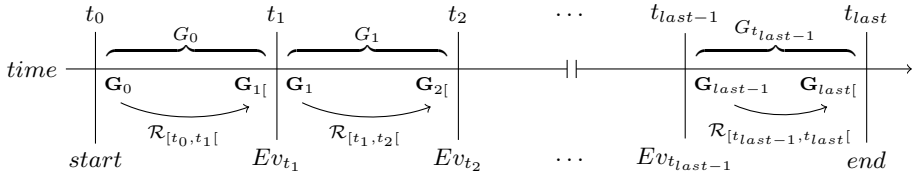


Fig. 4. Graph Relabellings and Evolving Graphs - Combined formalism.

3.2 Characterizing the topological assumptions of an algorithm

Below are some proposed methods and additional concepts to characterize the requirement of an algorithm in terms of topology dynamics. More precisely, we use the new combined formalism to define the notions of topology-related *necessary* and *sufficient* conditions, and discuss how they can be proved.

Objectives of an algorithm. Given an algorithm $\mathcal{A}$ and a labelled graph $\mathbf{G}$, the state one desires to reach can be given by a logic formula $\mathcal{P}$ on the labels of vertices and/or edges. In the case of the propagation scheme (Algorithm 1 Section 2.1), this could be that all nodes are informed,

$$\mathcal{P}_1(\mathbf{G}) = \forall v \in V(\mathbf{G}), \lambda(v) = I \;,$$

Now, if the objective (noted $\mathcal{O}$) is to *reach* such state at some point, then it can be simply expressed as $\mathcal{P}$ to be satisfied on the very last labelled graph of $\mathcal{G}$ (*e.g.* $\mathcal{O}_{\mathcal{A}_1} = \mathcal{P}_1(\mathbf{G}_{last})$ in the example). Whereas not covered in the examples, one could also consider algorithms whose objectives are to *maintain* a state (*e.g. self-stabilizing* algorithms), and express it for example as $\mathcal{O}_\mathcal{A} = \forall G_i \in S_G, \mathcal{P}(\mathbf{G}_{i+1[})$.

Necessary conditions. Given an algorithm $\mathcal{A}$, its objective $\mathcal{O}_\mathcal{A}$, an evolving graph $\mathcal{G}$ and an evolving graph property $\mathcal{C}_\mathcal{N}$. The property $\mathcal{C}_\mathcal{N}$ is a *(topology-related) necessary* condition for $\mathcal{O}_\mathcal{A}$ if and only if

$$\forall \mathcal{G}, \neg \mathcal{C}_\mathcal{N}(\mathcal{G}) \implies \neg \mathcal{O}_\mathcal{A}$$

Proving this result comes to prove that $\forall \mathcal{G}, \neg\, \mathcal{C}_\mathcal{N}(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid \mathcal{P}(\mathbf{G}_{last})$.

Sufficient conditions. Symmetrically, an evolving graph property $\mathcal{C}_\mathcal{S}$ is a *(topology-related) sufficient* condition for $\mathcal{A}$ if and only if

$$\forall \mathcal{G}, \mathcal{C}_\mathcal{S}(\mathcal{G}) \implies \mathcal{O}_\mathcal{A}$$

Proving this result comes to prove that $\forall \mathcal{G}, \mathcal{C}_\mathcal{S}(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}(\mathbf{G}_{last})$.

Discussion. No topology of any kind can guarantee, alone, that the nodes will effectively communicate and collaborate with each other. Hence, the characterization of any sufficient condition necessarily requires to make additional assumptions on the collaboration of nodes. We propose below a generic such assumption for the pairwise interaction model (depicted on Figure 2(c)). This assumption may or may not be considered as realistic depending on the expected rate of topological changes.

Progression Hypothesis 1 *(PH_1). For every given time interval $[t_i, t_{i+1}[$, with t_i in $\mathcal{S}_\mathbb{T}^{\setminus\{t_{last}\}}$, every vertex will be able to apply at least one relabelling rule with each of its neighbors, provided the rule preconditions are* already *satisfied at time t_i (and still satisfied at the time the rule is applied).*

4 First applications of the proposed framework

This section illustrates the proposed framework by the analysis of three basic algorithms, namely the propagation algorithm previously given, and two enumeration algorithms (one centralized, the other decentralized). The results obtained here are used in the next section to highlight some implications of this work.

4.1 Analysis of the propagation algorithm

We want to prove that the existence of a journey (*resp.* strict journey) between the emitter and every other node is a necessary (*resp.* sufficient) condition to achieve $\mathcal{O}_{\mathcal{A}_1}$ (complete the propagation). The point here is to show how these intuitive conditions can be *formally* established.

Condition 1 $\forall v \in V_\mathcal{G}^{\setminus\{emitter\}}, \exists \mathcal{J}_{(emitter,v)} \subseteq \mathcal{G}$
(It exists a journey between the emitter and every other vertex).

Lemma 1 $\forall v \in V_\mathcal{G} \mid \lambda_{t_0}(v) = N, \forall \sigma \in \mathbb{T}_{[t_0,t_{last}[}, \lambda_\sigma(v) = I \implies \exists u \in V_\mathcal{G}^{\setminus\{v\}}, \sigma' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma'}(u) = I, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
(If a non-emitter vertex has the information at some point, it implies the existence of an incoming journey from a vertex that had the information before)

Proof. $\forall v \in V_{\mathcal{G}} \mid \lambda_{t_0}(v) = N, \forall \sigma \in \mathbb{T}_{[t_0,t_{last}[}, (\lambda_\sigma(v) = I \implies \exists v' \in V_{\mathcal{G}}^{\setminus v} \mid r_{1(v',v)} \in \mathcal{R}_{\mathcal{A}_1[t_0,\sigma[}$) (If a non-emitter vertex has the information at some point, then it has applied rule r_1 with another vertex)
$\implies \exists v' \in V_{\mathcal{G}}^{\setminus v}, \sigma' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma'}(v') = I, (v', v) \in E(G_{pred(\sigma')})$
(An edge existed at a previous date between this vertex and a vertex labelled I)
By repetition, $\implies \exists v'' \in V_{\mathcal{G}}^{\setminus v}, \sigma'' \in \mathbb{T}_{[t_0,\sigma[} \mid \lambda_{\sigma''}(v'') = I, \exists \mathcal{J}_{(v'',v)} \subseteq \mathcal{G}$
(A journey existed from a node that had the information to the considered node) □

Proposition 1 *Condition 1 ($\mathcal{C}_1$) is a necessary condition on $\mathcal{G}$ to allow Algorithm 1 ($\mathcal{A}_1$) to reach its objective $\mathcal{O}_{\mathcal{A}_1}$.*

Proof. (using Lemma 1). Following from Lemma 1 and the initial states (I for the emitter, N for all other vertices), we have $\mathcal{O}_{\mathcal{A}_1} \implies \mathcal{C}_1$, and then $\neg\mathcal{C}_1 \implies \neg\mathcal{O}_{\mathcal{A}_1}$ □

Condition 2 $\forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{strict(emitter,v)} \subseteq \mathcal{G}$

Proposition 2 *Assuming the progression hypothesis (PH_1, defined in the previous section), Condition 2 ($\mathcal{C}_2$) is sufficient on $\mathcal{G}$ to guarantee that $\mathcal{A}_1$ will reach $\mathcal{O}_{\mathcal{A}_1}$.*

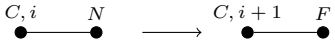
Proof. (1): By PH_1, $\forall t_i \in \mathcal{S}_{\mathbb{T}}^{\setminus(t_{last})}, \forall (u,u') \in E(G_i), (\lambda_{t_i}(u) = I \implies \lambda_{t_{i+1}}(u') = I)$
By iteration on (1): $\forall u, v \in V_{\mathcal{G}}, (\exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}) \implies (\lambda_{t_0}(u){=}I \implies \lambda_{t_{last}}(v){=}I)$
Now, because $\lambda_{t_0}(emitter) = I$, we have $\mathcal{C}_2(\mathcal{G}) \implies \forall X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}}, \mathcal{P}_1(\mathbf{G}_{last})$ □

4.2 Analysis of a centralized enumeration algorithm

Like the propagation algorithm, the distributed algorithm presented below assumes that one distinguished vertex is given a different initial state. This vertex, called the *counter*, is in charge of counting all the vertices it meets during the execution (its successive neighbors in the changing topology). Hence, the counter vertex has two labels (C, i), meaning that it is the counter (C), and that it has already counted i vertices (initially 1, *i.e.,* itself). The other vertices are labelled either F or N, depending on whether they have already been counted or not, respectively. The counting rule is given by r_1 in Algorithm 2, below.

Algorithm 2 Enumeration algorithm with a pre-selected counter.

initial states: $\{(C,1), N\}$ ($(C,1)$ for the counter, N for all other vertices)
alphabet: $\{C, N, F, \mathbb{N}^*\}$
rule r_1:



Objective of the algorithm. Under the assumption of a fixed number of vertices, the algorithm reaches the desired state when all vertices are counted, which corresponds to the fact that no more vertices are labelled N:

$$\mathcal{P}_2 = \forall v \in V(\mathbf{G}), \lambda(v) \neq N$$

The objective of Algorithm 2 is then to satisfy this property at the end of the execution ($\mathcal{O}_{\mathcal{A}_2} = \mathcal{P}_2(\mathbf{G}_{last})$). We want to prove here that the existence of an edge at some point of the execution between the *counter* node and every other node is a necessary and sufficient condition.

Condition 3 $\forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid (counter, v) \in E(G_i)$, *or equivalently with the notion of underlying graph,* $\forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, (counter, v) \in E_{\mathcal{G}}$

Proposition 3 *For a given evolving graph $\mathcal{G}$ representing the topological evolutions that take place during the execution of $\mathcal{A}_2$, Condition 3 ($\mathcal{C}_3$) is a* necessary *condition on $\mathcal{G}$ to allow $\mathcal{A}_2$ to reach its objective $\mathcal{O}_{\mathcal{A}_2}$.*

Proof. $\neg\mathcal{C}_3(\mathcal{G}) \implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid (counter, v) \notin E(G)$
$\implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid \forall t_i \in \mathcal{S}_{\mathbb{T}}^{\setminus\{t_{last}\}}, r_1(counter, v) \notin \mathcal{R}_{\mathcal{A}_2[t_i, t_{i+1}[}$
$\implies \exists v \in V_{\mathcal{G}}^{\setminus\{counter\}} \mid \forall X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}}, \lambda_{t_{last}}(v) = N$
$\implies \nexists X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}} \mid \mathcal{P}_2(\mathbf{G}_{last}) \implies \neg\mathcal{O}_{\mathcal{A}_2}$ □

Proposition 4 *Assuming the progression hypothesis (PH_1), $\mathcal{C}_3$ is also a sufficient condition on $\mathcal{G}$ to guarantee that $\mathcal{A}_2$ will reach its objective $\mathcal{O}_{\mathcal{A}_2}$.*

Proof. $\mathcal{C}_3(\mathcal{G}) \implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid (counter, v) \in E(G_i)$
by PH_1, $\implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \exists t_i \in \mathcal{S}_{\mathbb{T}} \mid r_1(counter, v) \in \mathcal{R}_{\mathcal{A}_2[t_i, t_{i+1}[}$
$\implies \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, \lambda_{t_{last}}(v) \neq N$
$\implies \forall X \in \mathcal{X}_{\mathcal{A}_2/\mathcal{G}}, \mathcal{P}_2(\mathbf{G}_{last}) \implies \mathcal{O}_{\mathcal{A}_2}$ □

4.3 Analysis of a decentralized enumeration algorithm

Contrary to the previous algorithm, Algorithm 3 below does not require a distinguished initial state for any vertex. Indeed, all vertices are initialized with the same labels $(C, 1)$, meaning that they are all initially counters that have already included themselves into the count. Then, depending on the topological evolutions, the counters opportunistically merge by pairs (rule r_1) in Algorithm $\mathcal{A}_3$. In the optimal case, at the end of the execution, only one node remains labelled C and its second label gives the total number of vertices in the graph. A similar counting principle has been used in [AAD+06] to monitor a flock of birds for fever, with the role of counters being played by sensors that have measured a high temperature level.

Algorithm 3 Decentralized enumeration algorithm.

initial states: $\{(C, 1)\}$ (for all vertices); *alphabet:* $\{C, F, \mathbb{N}^*\}$
rule r_1:

Objective of the algorithm Under the assumption of a fixed number of vertices, this algorithm reaches the desired state when exactly one vertex remains labelled C:

$$\mathcal{P}_3 = \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, \lambda(u) = C, \lambda(v) \neq C, \text{ and } \mathcal{O}_{\mathcal{A}_3} = \mathcal{P}_3(\mathbf{G}_{last})$$

For the sake of simplicity, we introduce one additional definition: the *destination set* of a vertex v in an evolving graph $\mathcal{G}$ is the set of all the vertices that can be reached from v by a journey, noted $\mathcal{D}est_{\mathcal{G}}(v)$. Note that $v \in \mathcal{D}est_{\mathcal{G}}(v)$ through an empty journey. We want to prove here that the existence of a journey from every vertex to at least one common destination vertex is a necessary condition for this algorithm.

Condition 4 $\exists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$

Lemma 2 $\forall u \in V_{\mathcal{G}} \mid \lambda_{t_i}(u) = C, \exists u' \in \mathcal{D}est_{\mathcal{G}}(u) \mid \lambda_{t_{j\geq i}}(u') = C$
(Whatever the C-labelled vertex considered at some point, there will be at a later point of the execution at least one vertex labelled C among its destination vertices)

Proof. (by contradiction). The application of r_1 is the only operation that can suppress a counter, while preserving the other counter in the pair. If Lemma 2 was false, then it would imply either that both counters have been discarded by r_1 at some point, or that the relabelling sequence has occurred from a C-labelled vertex towards a vertex that is outside of its destination set. Both are impossible. □

Proposition 5 *Condition 4 ($\mathcal{C}_4$) is* necessary *for $\mathcal{A}_3$ to reach its objective $\mathcal{O}_{\mathcal{A}_3}$.*

Proof. (using Lemma 2). $\neg\mathcal{C}_4(\mathcal{G}) \implies \nexists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$
(no vertices are destination for all the others).
$\implies \forall v \in V_{\mathcal{G}} \mid \lambda_{t_{last}}(v) = C, \exists u \in V_{\mathcal{G}} \mid v \notin \mathcal{D}est_{\mathcal{G}}(u)$
(Whatever the final counter, there is a vertex that could not reach it by a journey).
Now, thanks to Lemma 2, $\implies \forall v \in V_{\mathcal{G}} \mid \lambda_{t_{last}}(v) = C, \exists v' \in V_{\mathcal{G}}^{\setminus\{v\}} \mid \lambda_{t_{last}}(v') = C$
(There are at least two final counters).
$\implies \neg\mathcal{P}_3(\mathbf{G}_{last}) \implies \neg\mathcal{O}_{\mathcal{A}_3}$ □

The characterization of a sufficient condition for $\mathcal{A}_3$ is left open. We believe such a condition exists, but would be satisfied on a very few specific graphs.

5 Applications of the analysis results

This section presents some applications of the new framework. In particular, we show how the previously characterized conditions can be used to define evolving graph classes, some of which are included in others. This leads to a *de facto* classification of dynamic networks according to the algorithms they support. The relations between classes can be used in turn to compare algorithms on the basis of their topological requirements. Finally, we propose a method to check a given network trace for inclusion in each introduced class.

5.1 From conditions to graph classes

From $\mathcal{C}_1 = \forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{(emitter,v)} \subseteq \mathcal{G}$, we derive two classes of evolving graphs. $\mathcal{F}_1$ is the class in which at least one vertex can reach all the others by a journey. If an evolving graph does not belong to this class, then there is no chance for $\mathcal{A}_1$ to succeed whatever the initial emitter. $\mathcal{F}_2$ is the class where every vertex can reach all the others by a journey. If an evolving graph does not belong to this class, then at least one vertex, if chosen as an initial emitter, is guaranteed to fail to inform all the others using $\mathcal{A}_1$.

From $\mathcal{C}_2 = \forall v \in V_{\mathcal{G}}^{\setminus\{emitter\}}, \exists \mathcal{J}_{strict(emitter,v)} \subseteq \mathcal{G}$, we derive two classes of evolving graphs. $\mathcal{F}_3$ is the class in which at least one vertex can reach all the others by a strict journey. If an evolving graph belongs to this class, then there is at least one vertex that could, for sure, inform all the others using $\mathcal{A}_1$ (assuming the progression hypothesis). $\mathcal{F}_4$ is the class of evolving graphs in which every vertex can reach all the others by a strict journey. If an evolving graph belongs to this class, then the success of $\mathcal{A}_1$ is guaranteed for any vertex as initial emitter (again, if the progression hypothesis is assumed).

From $\mathcal{C}_3 = \forall v \in V_{\mathcal{G}}^{\setminus\{counter\}}, (counter, v) \in E_{\mathcal{G}}$, we derive two classes of graphs. $\mathcal{F}_5$ is the class of evolving graphs in which at least one vertex shares, at some point of the execution, an edge with every other vertex. If an evolving graph does not belong to this class, then there is no chance of success for $\mathcal{A}_2$, whatever the vertex chosen for counter. Here, if we assume the progression hypothesis, then $\mathcal{F}_5$ is also a class in which the success of the algorithm can be guaranteed for one specific vertex as counter. $\mathcal{F}_6$ is the class of evolving graphs in which every vertex shares an edge with every other vertex at some point of the execution. If an evolving graph does not belong to this class, then there exists at least one vertex for which, if it is chosen as the counter, the failure of $\mathcal{A}_2$ is guaranteed. Again, if we consider the progression hypothesis, then $\mathcal{F}_6$ becomes a class in which the success is guaranteed whatever the counter.

Finally, from $\mathcal{C}_4 = \exists v \in V_{\mathcal{G}} \mid \forall u \in V_{\mathcal{G}}, v \in \mathcal{D}est_{\mathcal{G}}(u)$, we derive the class $\mathcal{F}_7$, which is the class of graphs such that at least one vertex can be reached from all the others by a journey. If a graph does not belong to this class, then there is absolutely no chance of success for $\mathcal{A}_3$.

5.2 Relations between classes

Since *all* implies *at least one*, we have: $\mathcal{F}_2 \subseteq \mathcal{F}_1$, $\mathcal{F}_4 \subseteq \mathcal{F}_3$, and $\mathcal{F}_6 \subseteq \mathcal{F}_5$. Since a strict journey is a journey, we have: $\mathcal{F}_3 \subseteq \mathcal{F}_1$, and $\mathcal{F}_4 \subseteq \mathcal{F}_2$. Since an edge is a (strict) journey, we have: $\mathcal{F}_5 \subseteq \mathcal{F}_3$, $\mathcal{F}_6 \subseteq \mathcal{F}_4$, and $\mathcal{F}_5 \subseteq \mathcal{F}_7$. Finally, the existence of a journey between all pairs of vertices ($\mathcal{F}_2$) implies that each vertex can be reached by all the others, which implies in turn that at least one vertex can be reach by all the others ($\mathcal{F}_7$). We then have: $\mathcal{F}_2 \subseteq \mathcal{F}_7$. Although we have used here a non-strict inclusion ($\subseteq$), the inclusions described above are strict ($\subset$). This can be easily proved by finding for each inclusion a graph that belongs to the parent class but is outside the child class. Figure 5 summarizes all these relations.

$\mathcal{F}_1 : \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_2 : \forall u, v \in V_{\mathcal{G}}, \exists \mathcal{J}_{(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_3 : \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, \exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_4 : \forall u, v \in V_{\mathcal{G}}, \exists \mathcal{J}_{strict(u,v)} \subseteq \mathcal{G}$
$\mathcal{F}_5 : \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, (u,v) \in E_{\mathcal{G}}$
$\mathcal{F}_6 : \forall u, v \in V_{\mathcal{G}}, (u,v) \in E_{\mathcal{G}}$
$\mathcal{F}_7 : \exists u \in V_{\mathcal{G}} \mid \forall v \in V_{\mathcal{G}}^{\setminus\{u\}}, u \in \mathcal{D}est_{\mathcal{G}}(v)$

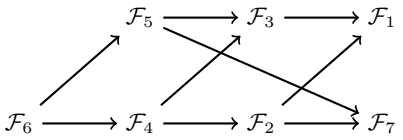


Fig. 5. A first classification of dynamic networks, based on evolving graph properties that result from the analysis of three distributed algorithms (*arrows denote inclusion*).

5.3 Comparison of algorithms according to topological assumptions

Let us consider the two enumeration algorithms given in Section 4. To have any chance of success, $\mathcal{A}_2$ requires the evolving graph to be in $\mathcal{F}_5$ (and a fortunate choice of counter) or in $\mathcal{F}_6$ (for possibly any vertex as counter). On the other hand, $\mathcal{A}_3$ requires the evolving graph to be in $\mathcal{F}_7$. Now, both classes $\mathcal{F}_5$ (directly) and $\mathcal{F}_6$ (transitively) are included in $\mathcal{F}_7$. As a consequence, there are some topological scenarios (*i.e.,* $\mathcal{G} \in {\mathcal{F}_7}^{\setminus\mathcal{F}_5}$) for which $\mathcal{A}_2$ has no chance of success, while $\mathcal{A}_3$ has some. Such observation allows to claim that $\mathcal{A}_3$ is more general than $\mathcal{A}_2$ with respect to its topological requirements. Hence, two algorithms can be fairly (and formally) compared on the basis of their topological requirements. In the particular case of these two enumeration algorithms, however, the claim could be balanced by the fact that a sufficient condition is known for $\mathcal{A}_2$, while no one is known for $\mathcal{A}_3$. The choice for the right algorithm may thus depend on the target mobility context: if this context is expected to induce topological scenarios in $\mathcal{F}_5$ or $\mathcal{F}_6$, then $\mathcal{A}_2$ could be preferred, otherwise $\mathcal{A}_3$ should be considered. More generally, it is however important to realize that a large gap may exist between *necessary* and *sufficient topology-related* conditions, and other topological properties (*resp.* evolving graph classes) could offer intermediate probabilities of success, which was not investigated for the given algorithms in this initial work.

5.4 Checking network traces for inclusion in the classes

We consider here the problem of checking automatically whether a given evolving graph belongs to one of the classes listed before. While having potentially a large scope of applications, this could allow in particular to help decide which algorithm is relevant to a given mobility context, by checking how the corresponding topological traces distribute over the classes. Below is a sketch of solution for each class met so far. The point is that all solutions can rely on common *static* graph properties, provided a few transformations. The *transitive closure* of an evolving graph $\mathcal{G}$ is the graph $H = (V, A_H)$, where $A_H = \{(v_i, v_j) : \exists \mathcal{J}_{(v_i,v_j)} \subseteq \mathcal{G})\}$. A transitive closure is by nature a *directed* graph, as illustrated in Figure 6, since journeys are oriented entities. As explained in [BF03], the computation of transitive closures can be done efficiently (in $O(|V_{\mathcal{G}}|.|E_{\mathcal{G}}|.(log|\mathcal{S}_{\mathbb{T}}|.log|V_{\mathcal{G}}|))$, by

building the tree of *shortest* journeys for each node in the network. We extend this notion to the case of strict journeys, with $H_{strict} = (V, A_{H_{strict}})$, where $A_{H_{strict}} = \{(v_i, v_j) : \exists \mathcal{J}_{strict(v_i,v_j)} \subseteq \mathcal{G})\}$.

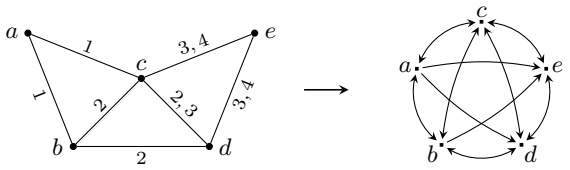


Fig. 6. Example of transitive closure of an evolving graph

Given an evolving graph $\mathcal{G}$, its underlying graph G, its transitive closure H, and the transitive closure of its *strict* journeys H_{strict}, the inclusion of $\mathcal{G}$ in each class can be checked as follows:

- $\mathcal{G} \in \mathcal{F}_1 \iff H$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_2 \iff H$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_3 \iff H_{strict}$ contains an out-dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_4 \iff H_{strict}$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_5 \iff G$ contains a dominating set of size 1.
- $\mathcal{G} \in \mathcal{F}_6 \iff G$ is a complete graph.
- $\mathcal{G} \in \mathcal{F}_7 \iff H$ contains an in-dominating set of size 1.

We expect most of the future classes to be possibly checked with similar approaches. This is however not a certainty.

6 Conclusion

This paper introduced a set of tools and methods dedicated to the analysis of distributed algorithms in dynamic networks. This new framework allows to characterize assumptions that a given algorithm requires in terms of topological evolution during its execution. It was illustrated by the analysis of three basic algorithms, and the analysis results were used to highlight potential implications of this work, including the possibility to compare algorithms on the basis of their topological requirements, and a sketch of classification of dynamic networks according to the corresponding properties. The problem of checking whether a given evolving graph belongs to the introduced classes was finally discussed.

Analyzing the requirement of algorithms is not a novel approach. It appears however that no proper transposition was previously done in the context of dynamic networks, where the usual practice is to liken dynamic topologies to static graphs. This is particularly striking in the recent field of *population protocols* [AAER07], where a common assumption is that all pairs of nodes interact repeatedly. In the light of the classification shown is this paper, such scenarios actually represent a subset of the most specific class among those discussed

(namely, $\mathcal{F}_6$). We think the framework proposed here could help characterize weaker assumptions for most population protocols.

The algorithms studied in this paper are simple. An interesting question for further research is whether the framework will scale to more complex algorithms, which remains unclear at this stage. We hope it could suit the study of common problems such as *electing*, *naming*, or *building spanning structures* (note that *electing* and *naming* may not have identical assumptions in a dynamic context). Another prospect is to investigate how intermediate properties could be explored between necessary and sufficient conditions, for example to guarantee a desired probability of success. Finally, as more properties are characterized and the classification grows, new insights may follow in the study of mobility models, based on checking generated traces for inclusion in the classes. Ultimately, this could answer questions like what kind of problems can be solved within a given mobility model, such as the well-known *random way point* model [BRS03], or in more realistic pedestrian and vehicular contexts.

References

[AAD+06] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[BF03] S. Bhadra and A. Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Proceedings of Adhoc-Now'03*, volume 2865 of *Lecture Notes in Computer Science*, pages 259–270, Montreal, October 2003.

[BRS03] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(3):257–269, 2003.

[CMZ06] J. Chalopin, Y. Métivier, and W. Zielonka. Local computations in graphs: The case of cellular edge local computations. *Fundamenta Informaticae*, 74(1):85–114, 2006.

[Fer04] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004. *A preliminary version appeared as* On models and algorithms for dynamic communication networks: The case for evolving graphs, *Algotel'02*, Meze, FR.

[GMMS02] E. Godard, Y. Métivier, M. Mosbah, and A. Sellami. Termination detection of distributed algorithms by graph relabelling systems. In *ICGT '02: Proceedings of the First International Conference on Graph Transformation*, pages 106–119, London, UK, 2002.

[LMS99] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In World Scientific Publishing, editor, *Handbook of graph grammars and computing by graph transformation*, volume III, Eds. H. Ehrig, H.J. Kreowski, U. Montanari and G. Rozenberg, pages 1–56, 1999.

[Lyn89] N. Lynch. A hundred impossibility proofs for distributed computing. In *PODC '89: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 1–28, New York, NY, USA, 1989. ACM.